May 26, 2015

Bhushan Kothari

Data Mining with spark

movie rating summarization

**Phase - 1**

1. First step that we do is read the file using a SparkContext object and its textFile() method. Thus after this, data is stored in a new rdd called r1.

***r1 = sc.textFile('Documents/ratings.txt')***

2. The file that we have read is ‘::’ separated file containing records which have following fields. **userID::movieID::rating::timestamp.**  Thus we split the file on ‘::’ by calling the map() method on rdd. Lambda represents the reference to the rdd records.

***r2 = r1.map(lambda x: (x.split('::')))***

3. After splitting the rdd, we just need the Movie Id and rating columns. Thus we can keep only the required columns using the map method and specifying the indexes. We need to calculate the average ratings for all movies, for which we need sum and total count of rating records. Thus we append 1 to each record. Map() method works as 1:1.

***r3 = r2.map(lambda x:(x[1],(x[2],1)))***

4. We need to typecast the ratings column to float so that we can perform the additions and division at later stages.

***r4 = r3.map(lambda x: (x[0], (float(x[1][0]),x[1][1])))***

5. Using reduceByKey() helps us in adding all the ratings corresponding to a particular movie also adding the 1’s appended in previous step helps us in obtaining the count of number of records having that movie.

***r5 = r4.reduceByKey(lambda x,y:(float(x[0])+float(y[0]), x[1]+y[1]))***

6. Right now we have output as movie ID, sum of ratings and count of records for that movie. Now we need to compute the average and it can be done using the map() method and dividing the summation by number of records. We now save the results into a file called ’average-ratings.txt’

***r6 = r5.map(lambda x:(x[0],float(x[1][0]/x[1][1])))***

***r6.saveAsTextFile('Documents/average-ratings.txt')***

**Phase - 2**

7. Filter() method in PySpark works only on Integers, thus we multiple the average by 100 so we can filter the data easily.

***r6 = r6.map(lambda x:(int(x[0]),int(x[1]\*100)))***

8. We create a function and pass the result of previous step (Movie ID, Avg\*100) to it. Function checks for the range of values and accordingly creates Bin ID for it. Suppose average ratings\*100 of movie fall between 0 & 50 then we replace the record by ‘0-0.5’, and so on. Thus we get output as bin ID.

***def createBin(avgRating):  
    bins=[]  
    strVal=str(avgRating[1])  
    if(int(strVal)>=0) and (int(strVal)<50):  
        avgRating="[0-0.5)"  
    elif(int(strVal)>=50) and (int(strVal)<100):  
        avgRating='[0.5-1)'  
    elif(int(strVal)>=100) and (int(strVal)<150):  
        avgRating='[1-1.5)'  
    elif(int(strVal)>=150) and (int(strVal)<200):  
        avgRating='[1.5-2)'  
    elif(int(strVal)>=200) and (int(strVal)<250):  
        avgRating='[2-2.5)'  
    elif(int(strVal)>=250) and (int(strVal)<300):  
        avgRating='[2.5-3)'  
    elif(int(strVal)>=300) and (int(strVal)<350):  
        avgRating='[3-3.5)'  
    elif(int(strVal)>=350) and (int(strVal)<400):  
        avgRating='[3.5-4)'  
    elif(int(strVal)>=400) and (int(strVal)<450):  
        avgRating='[4-4.5)'  
    elif(int(strVal)>=450) and (int(strVal)<501):  
        avgRating='[4.5-5]'  
    else:  
        avgRating = 0          
    return avgRating***

9. Call given to the function.

***r7 = r6.map(createBin)***

10. As we need to know the number of movies in each bins, we need to append 1 to the existing RDD.

***r8 = r7.map(lambda x:(x,1))***

11. Now we use the reduceByKey() method to sum the counts in bins.

***r9 = r8.reduceByKey(lambda x,y:(x+y))***

12. However, at this stage, we may have few ranges for which we don’t have any movies. Such ranges are ignored by the filter() command and thus we need to add such ranges to our final output to increase the output readability.

We create a new RDD allKeys which contains all possible bin ranges then we append 0 to all the ranges. Now we subtract our r9 RDD i.e. output obtained by reducing by keys from the allKeys RDD.

***allKeys = sc.parallelize(['[0-0.5)','[0.5-1)','[1-1.5)','[1.5-2)','[2-2.5)','[2.5-3)','[3-3.5)','[3.5-4)','[4-4.5)','[4.5-5]'])***

***allKeys = allKeys.map(lambda x:(x,0))  
missingKeys = allKeys.subtractByKey(r9)***

This gives us the ranges which did not have any movies in it and were excluded by the filter() command. ***missingKeys.collect()  
[('[0-0.5)', 0)]***

Now we add the excluded ranges to the r9 RDD. ***r10 = r9+missingKeys  
r10.collect()***

***[('[1.5-2)', 298), ('[3.5-4)', 3091), ('[4-4.5)', 462), ('[3-3.5)', 3684), ('[4.5-5]', 25), ('[2.5-3)', 2147), ('[2-2.5)', 919), ('[1-1.5)', 43), ('[0.5-1)', 8), ('[0-0.5)', 0)]***

After sortingbyKey() we save the file and then read the file and then read it and sort the keys and display the final output. ***rFinalOutput = r10.sortByKey()  
rFinalOutput.saveAsTextFile('Documents/ratings-summary.txt')***

**Phase - 3 *summary = sc.textFile('Documents/ratings-summary.txt')***

***summary. collect()   
[u"('[0-0.5)', 0)", u"('[1-1.5)', 43)", u"('[4-4.5)', 462)", u"('[1.5-2)', 298)", u"('[3.5-4)', 3091)", u"('[2.5-3)', 2147)", u"('[0.5-1)', 8)", u"('[2-2.5)', 919)", u"('[4.5-5]', 25)", u"('[3-3.5)', 3684)"]***

***summary = summary.map(lambda x:(x.split(',')))  
>>> summary.sortByKey().collect()***

***[(u"('[0-0.5)'", u' 0)'), (u"('[0.5-1)'", u' 8)'), (u"('[1-1.5)'", u' 43)'), (u"('[1.5-2)'", u' 298)'), (u"('[2-2.5)'", u' 919)'), (u"('[2.5-3)'", u' 2147)'), (u"('[3-3.5)'", u' 3684)'), (u"('[3.5-4)'", u' 3091)'), (u"('[4-4.5)'", u' 462)'), (u"('[4.5-5]'", u' 25)')]***

Thus to cross check our answer we can check the number of movies present in the dataset using the distinct() count and tallying this number with the totals of movies present in each of the bins. Bingo!! Numbers do match.

***r2.map(lambda x:(x[1])).distinct().count()***

*10677*